European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 7033

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	FEATURE INTERACTIONS IN TELECOMMUNICATIONS SYSTEMS III, 11 - 13 October 1995, AMSTERDAM NL, pages 157-172, XP000593331 BOSTRÖM ET AL.: "Feature interaction detection and resolution in the Delphi framework" -----		
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	27 March 1997	Lambley, S

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 833 526 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
01.04.1998 Bulletin 1998/14

(51) Int. Cl.[6]: H04Q 3/00

(21) Application number: 96307033.9

(22) Date of filing: 26.09.1996

(84) Designated Contracting States:
GB

(71) Applicant:
BRITISH TELECOMMUNICATIONS public limited company
London EC1A 7AJ (GB)

(72) Inventor:
The designation of the inventor has not yet been filed

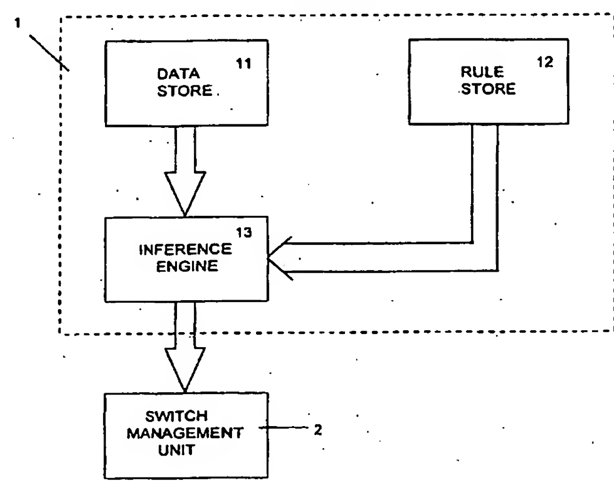
(74) Representative:
Wells, David et al
BT Group Legal Services
Intellectual Property Department,
8th Floor, Holborn Centre
120 Holborn
London EC1N 2TE (GB)

(54) Detecting service interactions in a telecommunications network

(57) A system for detecting interaction between different services on a telecommunications network includes a computer expert system. A data store in the expert system is programmed with data which represent attributes of service features. A rule store is programmed with rules which relate feature attributes to interaction behaviours. An inference engine is connected to the data store and to the rule store and processes the data and the rules to detect any interaction between the services.

The data in the data store may be arranged as sets of objects, each object in a set corresponding to a different state transition of the corresponding feature. The different objects may be given sequence numbers corresponding to the time sequence of execution of the feature. At least some of the rules may relate to these sequence numbers.

FIGURE 1



EP 0 833 526 A1

Description

The present invention relates to telecommunications networks, and in particular to the detection of undesirable interactions between different services running on a network.

Telecommunications networks are increasingly required to offer customers services in addition to basic call-handling. The development of novel network architectures, such as the IN (intelligent network) architecture, together with developments in computing platforms for telecommunications systems, make it potentially possible to offer customers a large portfolio of additional services to select from. However, as the number of services increases, and as services become available from independent service providers in addition to the network operator, then service feature interaction becomes a serious problem. It is often found that features offered by one service interact in an unwanted manner with features of other services. For example, a voice messaging service, such as BT's CallMinder service, may have as one of its standard features a behaviour such that incoming calls are diverted to the messaging service whenever the called line is busy. Another available service, Call Waiting, handles the same condition, namely the called number being busy, in an entirely different fashion. The Call Waiting service transmits an alert tone to the user and gives the user the option of interrupting the on-going call to speak to the new caller. It can be seen that if a customer wanted to subscribe to both services, then there is conflict between the service features which needs to be resolved. Otherwise, it would be necessary to bar the provisioning of both of these services to a customer, with a consequent loss in utility to the customer, and loss of revenue to the service provider.

Conventionally, during the planning of new services for a telecommunications network, attempts have been made to detect in advance any interaction problems by writing English-language specifications of the service features. Using these specifications a paper "walk through" of the services is then conducted, with the design engineer going step by step through the different services and spotting any interaction problems. This is a time-consuming procedure which can never be completely reliable, leaving the possibility that unforeseen interactions will occur when the service is deployed.

Some attempts have been made previously to automate the detection of interaction during the design phase. For example, International patent application WO95/22231 discloses a method of detecting service interactions which uses formal specifications of the additional services. The algorithm uses information that is specific to the services being tested which needs to be rewritten every time a change is made to one of the service features. Moreover, the approach adopted requires that a formal model should be prepared for every service feature which is handled. The preparation of such formal models is a difficult and time-consuming task requiring a high level of expertise.

According to a first aspect of the present invention, there is provided a system for detecting interaction between services running on a telecommunications network comprising:

a computer expert system including:

a) a data store programmed with data representing attributes of service features;
b) a rule store programmed with rules which relate feature attributes to interaction behaviours; and
c) an inference engine which is connected to the data store and to the rule store and which is arranged to process the data and the rules, thereby detecting any interaction between the services.

The present invention adopts a radically new approach to the detection of service feature interaction, through the use of an expert system. The traditional domain of use of expert systems is for diagnostic classification problems involving data which is essentially static. A well known example is the identification of a particular bacterium from a series of statements about its properties and appearance. In this domain, expert system technology has a proven track record in reproducing and sometimes surpassing human expert performance for the same problem. It is not however been thought possible hitherto to apply such techniques to the problem domain of the present invention. Feature interaction in a telecommunications network is an essentially time-related phenomen and so on the face of it not suitable for expert system techniques. The present inventors have found however that with an appropriate knowledge representation, expert systems can successfully be used for feature interaction detection. This provides a dramatic reduction in the time required to uncover service interworking problems, coupled with increased flexibility. The separation between the knowledge representation and the inference rules, means that changes or additions to the services can readily be assessed simply by making corresponding changes to the knowledge representation in the data store. In this way, the system is quickly able to detect any problems arising from the new features. By contrast with the prior art systems there is no need to repeat an entire "walk-through" from scratch.

The system of the present invention may run, for example, at a local switch in a telecommunications network to aid the detection and management of interaction problems as they occur. Alternatively, or in addition, the system may be used during the development of a new service to detect any interaction problems prior to the deployment of the service.

In the case of a system used at run-time in the network, then the output of the system may be fed to a control sys-

tem for modifying the behaviour of the network in order to remove or ameliorate the detected interaction problem. The control system may, for example, modify the stored profile for a customer in order to disable one or more service features. A more sophisticated control system might initiate a dialogue with the customer to allow the customer to determine a default behaviour for the network. For example, in the case of the Call Minder or call waiting services, the user might be given the option of selecting the Call Waiting response, that is the transmission of an alert tone, rather than the Call Minder response, that is the transfer of the incoming call to a messaging service.

Preferably the expert system data store includes a plurality of objects including, for each service feature which is represented in the data store, a set of objects corresponding to different respective state transitions of the feature.

The term "object" is used in this document in the sense of object oriented design/programming (OOD/OOP) methodologies.

As discussed in further detail below, the choice of an appropriate structure for organising the data in the data store is critical in maximising the efficiency of the interaction detection system. The inventors have found that the combination of the use of an object-based structure and a state transition representation of the service feature offers significant advantages both in efficiency of operation and in ease of development and modification of the detection system. The use of sets of objects representing different state transitions makes it possible to capture the characteristics of a service feature in a form which is well-adapted for processing by the inference engine.

Preferably each of the said set of state transition objects includes a sequence number corresponding to the position of the respective state transition in the sequence of execution of the feature and at least some of the rules in the rule store reason over the values of the sequence numbers. Preferably the objects are arranged in a hierarchy of superclasses and subclasses of the superclasses, and some of the rules reason over superclasses and others of the rules reason over subclasses.

The use of objects belonging to a hierarchy of classes, combined with rules which operate at different levels of the class hierarchy further increases the flexibility of the system. In particular, it ensures that when a new object is added to the data store, for example as the result of a modification to a service feature, there will already exist rules functioning at a higher level of the hierarchy which are immediately applicable to the new object, so that extensive modification of the rules is not required.

According to a second aspect of the present invention there is provided a method of detecting interaction between services running on a telecommunications network comprising:

programming a computer expert system with data representing attributes of service features and with rules relating feature attributes to interaction behaviours;
processing the said data and the said rules in an inference engine and detecting thereby any interaction between the said services.

According to a third aspect of the present invention, there is provided a method of operating a telecommunications network comprising:

programming a computer expert system with data representing attributes of service features and with rules relating feature attributes to interaction behaviours;
processing the said data and the said rules in an inference engine and detecting thereby any interaction between the said services; and
modifying the operation of the network when an interaction is detected.

Systems and methods embodying the present invention will now be described in further detail, by way of example only, with reference to the accompanying drawings in which:

Figure 1 is a schematic of a system for detecting service feature interaction;
Figure 2 is a schematic of a network having an IN architecture;
Figure 3 is a state transition diagram illustrating one example of a service feature;
Figure 4 is a class diagram showing the structure of the objects in the data store of the system of Figure 1;
Figure 5 shows the structure of the switch management unit;
Figure 6 shows a multiprocessor system for implementing the expert system of Figure 1.

A system for detecting and managing interactions between services running on a telecommunications network comprises an expert system 1 which in this example is connected to a switch management unit 2 within a service switching point (SSP) 3. As shown in Figure 2, the service switching point forms part of a telecommunications network employing an IN (intelligent network) architecture and including a further SSP 4, a service control point (SCP) 5 and an intelligent peripheral 6. The network, other than in the features described in further detail below, is conventional in

nature. For further details of the IN architecture reference is made to the paper by T W Abernethy & A C Munday, "Intelligent Networks, Standards and Services", BT Technol J, Vol. 13, No. 2, April 1995 and to the European Telecommunications Standards Institute Final Draft PRETS 300374-1, published July 1994, the contents of both of which are incorporated herein by reference.

The call control software within the switch management unit is structured as described in the paper by HM Blair, "Attacking Product Complexity: Broadband Call Control for Vision O.N.E" XIV International Switching Symposium, October 25-30 1992. Figure 5 shows the software structured into a chain of connection segments. In this software structure it is the responsibility of the User Transaction Segment (UTS) to invoke feature software and link it into the chain, based on customers' service data. In this example applicaton of the feature interaction expert system, the expert system forms part of the UTS. The expert system gets its input facts from the customer data and/or call progress signaling and the results of processing are used to decide whether to invoke feature software and if so, where in the call chain to link that software.

The expert system 1 includes a data store 11, a rule store 12 and an inference engine 13. In this example, the hardware for the expert system comprises a distributed processing system using Pentium microprocessors with access to local RAM and to mass storage devices. The data store 11 and rule store 12 are embodied in the storage devices and in the local RAM, and the inference engine 13 is provided by appropriate programming of the Pentium microprocessors.

Figure 6 shows in further detail the platform used in this example to support the expert system. Multiple Pentium CPU's are linked by a local bus to each other, and to the region of RAM. Data stored on a local hard disk is accessed via a SCSI interface. The multiprocessor system is implemented on a motherboard which is linked to other components of the switch by an FDDI optical fibre LAN. In this example, in order to facilitate the use of an object-oriented knowledge representation, the expert system is implemented using an expert systems shell available commercially from Neuron Data Inc. of Mountain View, CA as "NEXPERT OBJECT" (Trade Mark). This is an object-based expert systems implementation tool with facilities for implementation of rules in the C programming language. The implementation of the rules is based on first order predicate logic.

The expert system operates by applying rules to facts. The facts may have been input to the expert system either manually by the human user, or by a call control programme. Alternatively, the facts may have been inferred by previous applications of the rules. The evaluation of a rule assigns a truth value to the hypothesis of the rule, which represents some new fact about the domain. As rules may trigger other rules in their predicate actions, a set of rules comprises a network through which simple conclusions may be propagated to arrive at more complex results.

It is found to be important that an appropriate form of knowledge representation is used for the facts. This is particularly true for the problem domain of the present invention. As discussed in the introduction above, feature interaction is a characteristically time-related phenomenon, whereas expert systems have traditionally been applied to static diagnostic or classification problems. The preferred implementation of the present invention uses an object-based knowledge representation, in which the objects are derived from state transition models of the service features. A state transition model offers the functionality necessary to describe service features, provided that note is taken of the side effects of the transitions between the various states. It is the side effects that will lead to models becoming interdependent and hence interacting within the network. A telephony feature may make a state transition for a variety of reasons, including response to an event caused by a state transition for some other feature.

In the knowledge representation adopted in the present invention, the data store 11 is programmed with objects corresponding to the state transition of a finite state machine representing the behaviour of a telephony feature. As shown in Figure 4, the objects belong to classes which define a template for feature state transition objects.

It should be noted that the class does not relate to an instance of the feature acting on a particular call, but describes how the feature will behave depending on its context in a particular call. Rules may then be written concerning behaviour of members of these classes, without reference to actual values of call data.

Figure 3 is a state transition diagram representation of a service feature. In this example the service is an account code service, such as BT's chargecard service. Figure 4 is a class diagram showing the objects used to represent such a feature in a system embodying the present invention. Figure 4 uses the OMT diagram conventions set out in "Object Oriented Modeling & Design", Rumbaugh et al., Prentice Hall, ISBN 0-13-630054-5. As shown in the Figure, the account code service comprises four state transitions, referenced a-d. Within the knowledge base of the expert system, the feature is stored as an instance of the ServiceFeature class, comprising a name "ACCCode" and a set of four Feature Transition objects:

Transition a:

Sequence no. - 1

Trigger event -

event type - dialled digits
location - calling party
data - 145

Caused event -

event type - announcement
location- calling party
data - character string "A/C no. ?"

Transition b:

Sequence no. - 2

Trigger event -

event type - mid-call dialled digits
location - calling party
data - e.g. 56789 (valid)

Caused event -

event type - announcement
location- calling party
data - character string "Enter no. ?"

Transition c:

Sequence no. - 2

Trigger event -

event type - dialled digits
location - calling party
data - e.g. 34567 (not valid)

Caused event -

event type -clear call
location- general
data -

Transition d:

Sequence no. - 3

Trigger event -

event type - mid-call dialled digits
location - calling party
data - e.g. 01473 644668

Caused event -

event type -call event
location- general
data - e.g. 01473 644668

In developing the rules programmed in the rule store 12, use was made of a set of high level rules developed for this problem domain. These rules are grouped in terms of key words which collect the rules into concept groupings. The rules were then formalised and decomposed into smaller grains of knowledge to allow their implementation. Rules of small semantic weight use facts input to the experts system to extract simple conclusions. The simple conclusions are then forwarded through the rule network for use by one or more rules at a higher level. These larger semantic weight rules eventually form conclusions which correspond directly to the English language definitions of the top level rules.

As information is forwarded through the rule network, some re-use of the lower level rules is achieved. Simple components of knowledge which are useful in the implementation of one high level rule often recur in the conditions of another rule. The choice of rules used in the implementation affects the level of reusability which is achieved. An efficient implementation is arrived at by progressive refinement of the choice of low level rules used. The rule evaluation process moves up through the rule network from data to higher level conclusions. The successful evaluation of a rule often leads to a subset of the feature state transition objects existing within the knowledge base being associated with a conclusion. When this occurs, a class is created dynamically corresponding to the subset of feature states. Those of the higher level rules which make use of the associated hypotheses then generally operate on the subset of feature state transition objects rather than the complete set, thus progressively constraining the scope of this subset and eventually arriving at a smaller one showing some interaction.

Tables 2.1-2.5 show examples of the rules from the rule store 12. It can be seen that hypothesis of low semantic weight such as "trigger location compatible" lead to hypothesis with semantic weight corresponding to the original English language rules such as "triggering conflict". Table 2.4 shows a rule operating on the feature transition attribute "sequence number" to infer that a particular service feature is "persistent" (it remains associated with a call after its initial actions when triggered). Table 2.5 shows a rule operating on objects belonging to the super-class "run time event" to infer that a particular service feature initiates a new call as part of its actions.

In use, when operation of the inference engine results in the hypothesis of rule 2.3 having the value "TRUE" then this result may be acted on by the switch management unit 2 to modify or inhibit one or more of the features running on the switch, so as to resolve the conflict arising from the feature interaction problem. For the particular call control software architecture illustrated in Figure 5, this involves the User Transaction Segment (UTS) either not invoking the Feature Segement (FS) corresponding to one of the features concerned or modifying the positions in the call chain of the relevant FSs.

TABLE 2.1

If < |feature_transition| >.triggger_event_location equals "remote"

And < |feature_transition| >.trigger_event_location equals

next_feature_transition.trigger_event_location

Then trigger_locations_compatible

Action Create Object < |feature_transition| > location_compatible-transition|

TABLE 2.2

If trigger_locations_compatible

And < |location__compatible_transition| >.trigger_event_location equals "remote"

And < |location__compatible_transition| >.dest_line_state equals next_feature_transition.

dest_line_state

Then line_transition_compatible

Action Create Object < |location__compatible_transition| > |compatible__transition|

TABLE 2.3

If line_states_compatible

And < |compatible_transition| >.trigger_event_type equals next_feature_transition.trigger_event_type

And < |compatible_transition| >.local_line_state equals next_feature_transition.local_state

Then triggering_conflict

TABLE 2.4

Rule using the Feature_transition attribute "Sequence number"
If < |service feature| >.< |feature_transitions| >.sequence_number.1
Then persistent feature
Action Ceate Object < |service feature| > |persistent_features|

TABLE 2.5

Rule using the "Run time event" class
If < |service feature| >.< |run time_event| >.event_type equals "new call"
The multiple calls feature
Action Create Object < |service_feature| > |multiple_calls_features|

Appendix A - Example expert system

The following definitions document an expert system embodying the invention. The first section describes the expert system in terms of the class structures of the knowledge base, the static objects in the knowledge base and the rules which act on objects in the knowledge base. The second section is a snap-shot of the objects in the expert system after details of several service features have been added to it. The definitions are printed out from the ***Nexpert Object***™ tool in the format as documented by the Nexpert object reference manual (January 1991), part number Man-10-400-01.

A basic undertanding of object oriented principles as well as expert system basics are assumed. Simple types used are Boolean (B), Integer (I) and String (S).

Expert system definition

Class definitions

NAME : announce_history

NAME : cascade_attatch
PROPERTIES :
feature_name = (S) Unknown

NAME : compatible_states
PROPERTIES :
local_line_state = (S) Unknown
trigger_event_type = (S) Unknown

NAME : conflicting_local_announcement

NAME : conflicting_remote_announcement

NAME : control_codes
PROPERTIES :
cascade_attachments = (S) Unknown
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown
dest_line1_state = (S) Unknown
dest_line2_state = (S) Unknown
feature_name = (S) Unknown
local_line_state = (S) Unknown
location_attachments = (S) Unknown
rank = (I) Unknown

sequence_number = (I) Unknown
trigger_event_location = (S) Unknown
trigger_event_type = (S) Unknown

NAME : feature_states
SUBCLASSES :
states_compatible_return
returned_lsc
lsc_out
working_feature_states
triggering_conflicts
control_codes
PROPERTIES :
cascade_attachments = (S) Unknown

NAME : feature_states.cascade_attachments
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown
dest_line1_state = (S) Unknown
dest_line2_state = (S) Unknown
feature_name = (S) Unknown
local_line_state = (S) Unknown
location_attachments = (S) Unknown
rank = (I) Unknown
sequence_number = (I) Unknown
trigger_event_location = (S) Unknown
trigger_event_type = (S) Unknown

NAME : location_compatible_states
PROPERTIES :
dest_line1_state = (S) Unknown
local_line_state = (S) Unknown
trigger_event_location = (S) Unknown

NAME : lsc_out
PROPERTIES :
cascade_attachments = (S) Unknown
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown
dest_line1_state = (S) Unknown
dest_line2_state = (S) Unknown
feature_name = (S) Unknown
local_line_state = (S) Unknown
location_attachments = (S) Unknown

rank = (I) Unknown
sequence_number = (I) Unknown
trigger_event_location = (S) Unknown
trigger_event_type = (S) Unknown

NAME : names_attached
PROPERTIES :
feature_name = (S) Unknown
trigger_event_location = (S) Unknown

NAME : old_value

NAME : old_values
PROPERTIES :
feature_name = (S) Unknown

NAME : possible_features
PROPERTIES :
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown
feature_name = (S) Unknown
trigger_event_location = (S) Unknown

NAME : returned_lsc
PROPERTIES :
cascade_attachments = (S) Unknown
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown
dest_line1_state = (S) Unknown
dest_line2_state = (S) Unknown
feature_name = (S) Unknown
local_line_state = (S) Unknown
location_attachments = (S) Unknown
rank = (I) Unknown
sequence_number = (I) Unknown
trigger_event_location = (S) Unknown
trigger_event_type = (S) Unknown

NAME : states_compatible_return
PROPERTIES :
cascade_attachments = (S) Unknown
caused_local_event = (S) Unknown
caused_remote_event1 = (S) Unknown
caused_remote_event2 = (S) Unknown

dest_line1_state =	(S) Unknown
dest_line2_state =	(S) Unknown
feature_name =	(S) Unknown
local_line_state =	(S) Unknown
location_attachments =	(S) Unknown
rank =	(I) Unknown
sequence_number =	(I) Unknown
trigger_event_location =	(S) Unknown
trigger_event_type =	(S) Unknown

NAME : triggering_conflicts
PROPERTIES :

cascade_attachments =	(S) Unknown
caused_local_event =	(S) Unknown
caused_remote_event1 =	(S) Unknown
caused_remote_event2 =	(S) Unknown
dest_line1_state =	(S) Unknown
dest_line2_state =	(S) Unknown
feature_name =	(S) Unknown
local_line_state =	(S) Unknown
location_attachments =	(S) Unknown
rank =	(I) Unknown
sequence_number =	(I) Unknown
trigger_event_location =	(S) Unknown
trigger_event_type =	(S) Unknown

NAME : working_feature_states
PROPERTIES :

cascade_attachments =	(S) Unknown
caused_local_event =	(S) Unknown
caused_remote_event1 =	(S) Unknown
caused_remote_event2 =	(S) Unknown
dest_line1_state =	(S) Unknown
dest_line2_state =	(S) Unknown
feature_name =	(S) Unknown
local_line_state =	(S) Unknown
location_attachments =	(S) Unknown
rank =	(I) Unknown
sequence_number =	(I) Unknown
trigger_event_location =	(S) Unknown
trigger_event_type =	(S) Unknown

Properties definitions
NAME : cascade_attachments
TYPE : String

NAME : caused_local_event
TYPE : String

NAME : caused_remote_event1
TYPE : String

NAME : caused_remote_event2
TYPE : String

NAME : dest_line1_state
TYPE : String

NAME : dest_line2_state
TYPE : String

NAME : feature_name
TYPE : String

NAME : local_line_state
TYPE : String

NAME : location_attachments
TYPE : String

NAME : rank
TYPE : Integer

NAME : returnval
TYPE : Integer

NAME : sequence_number
TYPE : Integer

NAME : trigger_event_location
TYPE : String

NAME : trigger_event_type
TYPE : String

NAME : Value
TYPE : Special

Rules definitions
RULE : Rule 29
If
there is evidence of incremental_state_development

And there is no evidence of cascade_triggering_exists
And <|feature_states|>.feature_name is equal to next_feature_state.feature_name
And (next_feature_state.sequence_number-<|feature_states|>.sequence_number) is precisely equal to 1
And Execute "RankList"(@ATOMID=<|feature_states|>;@STRING="@RANKBY=cascade_attachments,@RANKSET=rank,@INCREASING";)
And <|feature_states|>.rank is precisely equal to 1
And next_feature_state.cascade_attachments is not equal to "dummy"
And next_feature_state.location_attachments is not equal to "dummy"
And Execute "add_cascade"(@ATOMID=next_feature_state,<|feature_states|>;)
Then cascade_history
is confirmed.

RULE : Rule 2
If
there is evidence of incremental_state_development .
And there is no evidence of cascade_triggering_exists
And <|feature_states|>.feature_name is equal to next_feature_state.feature_name
And (next_feature_state.sequence_number-<|feature_states|>.sequence_number) is precisely equal to 1
And Execute "RankList"(@ATOMID=<|feature_states|>;@STRING="@RANKBY=cascade_attachments,@RANKSET=rank,@INCREASING";)
And <|feature_states|>.rank is precisely equal to 1
And next_feature_state.cascade_attachments is not equal to "dummy"
And next_feature_state.location_attachments is not equal to "dummy"
And Execute "add_cascade"(@ATOMID=next_feature_state,<|feature_states|>;)
Then cascade_history
is confirmed.

RULE : Rule 1
If
there is evidence of incremental_state_development
And there is evidence of cascade_triggering_exists
And <|feature_states|>.feature_name is equal to next_feature_state.feature_name
And (next_feature_state.sequence_number-<|feature_states|>.sequence_number) is precisely equal to 1
And Execute "RankList"(@ATOMID=<|feature_states|>;@STRING="@RANKBY=cascade_attachments,@RANKSET=rank,@INCREASING";)
And <|feature_states|>.rank is precisely equal to 1
And next_feature_state.cascade_attachments is not equal to "dummy" .
And next_feature_state.location_attachments is not equal to "dummy"
And Execute "add_cascade"(@ATOMID=next_feature_state,<|feature_states|>,<|lsc_out|>;)
Then cascade_history
is confirmed.

RULE : Rule 6

If

<|working_feature_states|>.trigger_event_type is equal to next_feature_state.caused_remote_event1
And <|working_feature_states|>.trigger_event_location is precisely equal to "remote"
And next_feature_state.dest_line1_state is not equal to "police"
And Delete Object |returned_lsc|
And Delete Object |lsc_out|
And Create Object <|working_feature_states|> |lsc_out|
And Execute
"line_states_compatible"(@ATOMID=|lsc_out|,next_feature_state;@STRING="local_line_state,dest_line1_state";)
Then cascade_triggering_exists
is confirmed.

RULE : Rule 5
If

<|working_feature_states|>.trigger_event_type is equal to next_feature_state.caused_local_event
And <|working_feature_states|>.trigger_event_location is precisely equal to "remote"
And next_feature_state.local_line_state is not equal to "police"
And Delete Object |lsc_out|
And Delete Object |returned_lsc|
And Create Object <|working_feature_states|> |lsc_out|
And Execute
"line_states_compatible"(@ATOMID=|lsc_out|,next_feature_state;@STRING="dest_line1_state,local_line_state";)
Then cascade_triggering_exists
is confirmed.

RULE : Rule 4
If

<|working_feature_states>.trigger_event_type is equal to next_feature_state.caused_remote_event2
And <|working_feature_states|>.trigger_event_location is precisely equal to "local"
And next_feature_state.dest_line2_state is not equal to "police"
And Delete Object |lsc_out|
And Delete Object |returned_lsc|
And Create Object <|working_feature_states|> |lsc_out|
And Execute
"line_states_compatible"(@ATOMID=|lsc_out|,next_feature_state;@STRING="local_line_state,dest_line2_state";)
Then cascade_triggering_exists
is confirmed.

RULE : Rule 3
If

<|working_feature_states|>.trigger_event_type is equal to next_feature_state.caused_remote_event1
And <|working_feature_states|>.trigger_event_location is precisely equal to "local"
And next_feature_state.dest_line1_state is not equal to "police"
And Delete Object |returned_lsc|
And Delete Object |lsc_out|
And Create Object <|working_feature_states|> |lsc_out|

And Execute
"line_states_compatible"(@ATOMID=|lsc_out|,next_feature_state;@STRING="local_line_state.dest_line1_state";)
Then cascade_triggering_exists
is confirmed.

RULE : Rule 7
If
there is evidence of possible_features_listed
And <|feature_states|>.trigger_event_type is precisely equal to "control code"
And <|feature_states|>.feature_name is equal to <|possible_features|>.feature_name
And <|possible_features|>.trigger_event_location is precisely equal to "local"
And Create Object <|feature_states|> |control_codes|
Then control_code_history
is confirmed.

RULE : Rule 8
If
<|feature_states|>.feature_name is equal to next_feature_state.feature_name
And <|feature_states|>.sequence_number is equal to next_feature_state.sequence_number
Then duplicate_state
is confirmed.

RULE : Rule 9
If
<|feature_states|>.feature_name is equal to next_feature_state.feature_name
And <|feature_states|>.caused_local_event is precisely equal to "announcement"
And Create Object <|feature_states|> |announce_history|
Then history_built
is confirmed.

RULE : Rule 11
If
<|feature_states|>.feature_name is equal to next_feature_state.feature_name
And Delete Object |working_feature_states|
And Create Object <<|feature_states|>> |working_feature_states|
Then incremental_state_development
is confirmed.

RULE : Rule 10
If
<|feature_states|>.feature_name is equal to next_feature_state.feature_name
And Delete Object |working_feature_states|
And Create Object <<|feature_states|>> |working_feature_states|
And (<|feature_states|>.sequence_number-next_feature_state.sequence_number) is greater than or equal to 0
And Delete Object <|feature_states|> |working_feature_states|
Then incremental_state_development

is confirmed.

RULE : Rule 12
If
there is evidence of trigger_locations_compatible
And next_feature_state.local_line_state is not equal to "police"
And next_feature_state.dest_line1_state is not equal to "police"
And Execute "feature_states_compatible"(@ATOMID=|location_compatible_states|,next_feature_state;)
Then line_states_ok
is confirmed.

RULE : Rule 15
If
there is evidence of possible_features_listed
And <|possible_features|>.caused_local_event is precisely equal to "announcement"
And <|names_attached|>.feature_name is equal to <|possible_features|>.feature_name
And <|names_attached|>.trigger_event_location is precisely equal to "remote"
And <|possible_features|>.feature_name is equal to <|names_attached|>.feature_name
And Create Object <|possible_features|> |conflicting_local_announcement|
Then local_announcement_conflict
is confirmed.

RULE : Rule 14
If
there is evidence of possible_features_listed
And <|possible_features|>.caused_remote_event1 is precisely equal to "announcement"
And <|names_attached|>.feature_name is equal to <|possible_features|>.feature_name
And <|names_attached|>.trigger_event_location is precisely equal to "local"
And <|possible_features|>.feature_name is equal to <|names_attached|>.feature_name
And Create Object <|possible_features|> |conflicting_local_announcement|
Then local_announcement_conflict
is confirmed.

RULE : Rule 13
If
there is evidence of possible_features_listed
And <|possible_features|>.caused_remote_event2 is precisely equal to "announcement"
And <|names_attached|>.feature_name is equal to <|possible_features|>.feature_name
And <|names_attached|>.trigger_event_location is precisely equal to "local"
And <|possible_features|>.feature_name is equal to <|names_attached|>.feature_name
And Create Object <|possible_features|> |conflicting_local_announcement|
Then local_announcement_conflict
is confirmed.

RULE : Rule 16
If

```
        there is no evidence of incremental_state_development
        And Delete Object |working_feature_states|
        And Create Object |feature_states| |working_feature_states|
Then new_feature_development
        is confirmed.

RULE : Rule 17
If
        there is evidence of cascade_history
        And Execute "mv_length"(@ATOMID=next_feature_state.cascade_attachments.temp.returnval:)
        And Execute "create_objs"(@ATOMID=temp.returnval,|names_attached|;@STRING="attached_name";)
        And Execute
"GetMultiValue"(@ATOMID=next_feature_state.cascade_attachments,<|names_attached|>.feature_name;@STRING="@STRAT=SETFWRD";)
        And Execute
"GetMultiValue"(@ATOMID=next_feature_state.location_attachments,<|names_attached|>.trigger_event_location:)
        And <|feature_states|>.feature_name is equal to <|names_attached|>.feature_name
        And Create Object <|feature_states|> |possible_features|
Then possible_features_listed
        is confirmed.

RULE : Rule 19
If
        there is evidence of possible_features_listed
        And <|possible_features|>.caused_remote_event2 is precisely equal to "announcement"
        And <|names_attached|>.feature_name is equal to <|possible_features|>.feature_name
        And <|names_attached|>.trigger_event_location is precisely equal to "local"
        And <|possible_features|>.feature_name is equal to <|names_attached|>.feature_name
        And Create Object <|possible_features|> |conflicting_remote_announcement|
Then remote_announcement_conflict
        is confirmed.

RULE : Rule 18
If
        there is evidence of possible_features_listed
        And <|possible_features|>.caused_remote_event1 is precisely equal to "announcement"
        And <|names_attached|>.feature_name is equal to <|possible_features|>.feature_name
        And <|names_attached|>.trigger_event_location is precisely equal to "local"
        And <|possible_features|>.feature_name is equal to <|names_attached|>.feature_name
        And Create Object <|possible_features|> |conflicting_remote_announcement|
Then remote_announcement_conflict
        is confirmed.

RULE : Rule 20
If
        Execute "report"()
```

Then report_generated

is confirmed.

RULE : Rule 21

If

Execute "SetMultiValue"(@ATOMID=next_feature_state.cascade_attachments;@STRING="@ADD=Call Waiting, Call Diversion, Call Divert on Busy,@NODUPLICATE,@COMP=STRING";)

And Execute "SetMultiValue"(@ATOMID=next_feature_state.location_attachments;@STRING="@ADD=local, local, remote,@DUPLICATE,@COMP=STRING";)

And Execute "add_cascade"(@ATOMID=next_feature_state;@STRING="Call Back When Free, local";)

Then test

is confirmed.

RULE : Rule 22

If

next_feature_state.dest_line1_state is not equal to "police"

And Execute "line_states_compatible"(@ATOMID=|feature_states|,next_feature_state;@STRING="local_line_state,dest_line1_state";)

Then test2

is confirmed.

RULE : Rule 24

If

<|feature_states|>.trigger_event_location is precisely equal to "local"

And <|feature_states|>.trigger_event_location is equal to next_feature_state.trigger_event_location

Then trigger_locations_compatible

is confirmed.

And Create Object <|feature_states|> |location_compatible_states|

RULE : Rule 23

If

<|feature_states|>.trigger_event_location is precisely equal to "remote"

And <|feature_states|>.trigger_event_location is equal to next_feature_state.trigger_event_location

Then trigger_locations_compatible

is confirmed.

And Create Object <|feature_states|> |location_compatible_states|

RULE : Rule 28

If

there is evidence of line_states_ok

And <|states_compatible_return|>.trigger_event_type is equal to next_feature_state.trigger_event_type

And Create Object <|states_compatible_return|> |triggering_conflicts|

Then triggering_conflict

is confirmed.

RULE : Rule 27

If

<|feature_states|>.trigger_event_type is equal to next_feature_state.caused_local_event

And <|feature_states|>.local_line_state is equal to next_feature_state.local_line_state

And Create Object <|feature_states|> |triggering_conflicts|

Then triggering_conflict

is confirmed.

RULE : Rule 26

If

<|feature_states|>.trigger_event_type is equal to next_feature_state.caused_remote_event1

And <|feature_states|>.dest_line1_state is equal to next_feature_state.dest_line1_state

And Create Object <|feature_states|> |triggering_conflicts|

Then triggering_conflict

is confirmed.

RULE : Rule 25

If

<|feature_states|>.trigger_event_type is equal to next_feature_state.caused_remote_event2

And <|feature_states|>.dest_line1_state is equal to next_feature_state.dest_line1_state

And Create Object <|feature_states|> |triggering_conflicts|

Then triggering_conflict

is confirmed.

List of hypotheses

cascade_history:	Unknown
cascade_triggering_exists:	Unknown
• control_code_history:	Unknown
• duplicate_state:	Unknown
• history_built:	Unknown
incremental_state_development:	Unknown
line_states_ok:	Unknown
• local_announcement_conflict:	Unknown
• new_feature_development:	Unknown
possible_features_listed:	Unknown
• remote_announcement_conflict:	Unknown
• report_generated:	Unknown
• test:	Unknown
• test2:	Unknown
trigger_locations_compatible:	Unknown

• triggering_conflict: *Unknown*

Snap-shot of objects in the expert system aft r details of sev ral service features have been added

Objects describing the *Call back when free* feature

NAME : CBWF1
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CBWF1.cascade_attachments
caused_local_event = (S) announcement

NAME : CBWF1.caused_local_event
caused_remote_event1 = (S) none

NAME : CBWF1.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CBWF1.caused_remote_event2
dest_line1_state = (S) any

NAME : CBWF1.dest_line1_state
dest_line2_state = (S) any

NAME : CBWF1.dest_line2_state
feature_name = (S) Call Back When Free

NAME : CBWF1.feature_name
local_line_state = (S) idle

NAME : CBWF1.local_line_state
location_attachments = (S) local

NAME : CBWF1.location_attachments
rank = (I) Unknown
sequence_number = (I) 1

NAME : CBWF1.sequence_number
trigger_event_location = (S) local

NAME : CBWF1.trigger_event_location
trigger_event_type = (S) control_code

NAME : CBWF1.trigger_event_type

NAME : CBWF2
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) Call Diversion,Call Waiting,Call Divert On Busy

NAME : CBWF2.cascade_attachments
caused_local_event = (S) seize

NAME : CBWF2.caused_local_event
caused_remote_event1 = (S) seize

NAME : CBWF2.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CBWF2.caused_remote_event2
dest_line1_state = (S) busy

NAME : CBWF2.dest_line1_state
dest_line2_state = (S) any

NAME : CBWF2.dest_line2_state
feature_name = (S) Call Back When Free

NAME : CBWF2.feature_name
local_line_state = (S) any

NAME : CBWF2.local_line_state
location_attachments = (S) local,local,local

NAME : CBWF2.location_attachments
rank = (I) Unknown
sequence_number = (I) 2

NAME : CBWF2.sequence_number
trigger_event_location = (S) remote

NAME : CBWF2.trigger_event_location
trigger_event_type = (S) clear

NAME : CBWF2.trigger_event_type

Objects describing the *Call diversion* feature
NAME : CD1

CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CD1.cascade_attachments
caused_local_event = (S) announcement

NAME : CD1.caused_local_event
caused_remote_event1 = (S) none

NAME : CD1.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CD1.caused_remote_event2
dest_line1_state = (S) any

NAME : CD1.dest_line1_state
dest_line2_state = (S) any

NAME : CD1.dest_line2_state
feature_name = (S) Call Diversion

NAME : CD1.feature_name
local_line_state = (S) any

NAME : CD1.local_line_state
location_attachments = (S) local

NAME : CD1.location_attachments
rank = (I) Unknown
sequence_number = (I) 1

NAME : CD1.sequence_number
trigger_event_location = (S) local

NAME : CD1.trigger_event_location
trigger_event_type = (S) control code

NAME : CD1.trigger_event_type

NAME : CD2
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) Call Waiting, Call Diversion

NAME : CD2.cascade_attachments
caused_local_event = (S) none

NAME : CD2.caused_local_event
caused_remote_event1 = (S) seize

NAME : CD2.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CD2.caused_remote_event2
dest_line1_state = (S) any

NAME : CD2.dest_line1_state
dest_line2_state = (S) any

NAME : CD2.dest_line2_state
feature_name = (S) Call Diversion

NAME : CD2.feature_name
local_line_state = (S) any

NAME : CD2.local_line_state
location_attachments = (S) local

NAME : CD2.location_attachments
rank = (I) Unknown
sequence_number = (I) 2

NAME : CD2.sequence_number
trigger_event_location = (S) local

NAME : CD2.trigger_event_location
trigger_event_type = (S) seize

NAME : CD2.trigger_event_type

NAME : CD3
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CD3.cascade_attachments
caused_local_event = (S) none

NAME : CD3.caused_local_event
caused_remote_event1 = (S) clear

NAME : CD3.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CD3.caused_remote_event2
dest_line1_state = (S) busy

NAME : CD3.dest_line1_state
dest_line2_state = (S) any

NAME : CD3.dest_line2_state
feature_name = (S) Call Diversion

NAME : CD3.feature_name
local_line_state = (S) any

NAME : CD3.local_line_state
location_attachments = (S) local

NAME : CD3.location_attachments
rank = (I) Unknown
sequence_number = (I) 3

NAME : CD3.sequence_number
trigger_event_location = (S) local

NAME : CD3.trigger_event_location
trigger_event_type = (S) clear

NAME : CD3.trigger_event_type

Objects describing the *Call waiting* feature

NAME : CW1
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CW1.cascade_attachments
caused_local_event = (S) announcement

NAME : CW1.caused_local_event
caused_remote_event1 = (S) none

NAME : CW1.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CW1.caused_remote_event2
dest_line1_state = (S) any

NAME : CW1.dest_line1_state
dest_line2_state = (S) any

NAME : CW1.dest_line2_state
feature_name = (S) Call Waiting

NAME : CW1.feature_name
local_line_state = (S) idle

NAME : CW1.local_line_state
location_attachments = (S) local

NAME : CW1.location_attachments
rank = (I) Unknown
sequence_number = (I) 1

NAME : CW1.sequence_number
trigger_event_location = (S) local

NAME : CW1.trigger_event_location
trigger_event_type = (S) control code

NAME : CW1.trigger_event_type

NAME : CW2
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CW2.cascade_attachments
caused_local_event = (S) beep

NAME : CW2.caused_local_event
caused_remote_event1 = (S) none

NAME : CW2.caused_remote_event1
caused_remote_event2 = (S) announcement

NAME : CW2.caused_remote_event2
dest_line1_state = (S) call in progress

NAME : CW2.dest_line1_state
dest_line2_state = (S) dialling

NAME : CW2.dest_line2_state
feature_name = (S) Call Waiting

NAME : CW2.feature_name
local_line_state = (S) call in progress

NAME : CW2.local_line_state
location_attachments = (S) local

NAME : CW2.location_attachments
rank = (I) Unknown
sequence_number = (I) 2

NAME : CW2.sequence_number
trigger_event_location = (S) local

NAME : CW2.trigger_event_location
trigger_event_type = (S) seize

NAME : CW2.trigger_event_type

NAME : CW3a
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CW3a.cascade_attachments
caused_local_event = (S) announcement

NAME : CW3a.caused_local_event
caused_remote_event1 = (S) clear

NAME : CW3a.caused_remote_event1
caused_remote_event2 = (S) none

NAME : CW3a.caused_remote_event2
dest_line1_state = (S) call in progress

NAME : CW3a.dest_line1_state

dest_line2_state = (S) call in progress

NAME : CW3a.dest_line2_state
feature_name = (S) Call Waiting

NAME : CW3a.feature_name
local_line_state = (S) call in progress

NAME : CW3a.local_line_state
location_attachments = (S) local

NAME : CW3a.location_attachments
rank = (I) Unknown
sequence_number = (I) 3

NAME : CW3a.sequence_number
trigger_event_location = (S) local

NAME : CW3a.trigger_event_location
trigger_event_type = (S) control code

NAME : CW3a.trigger_event_type

NAME : CW3b
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : CW3b.cascade_attachments
caused_local_event = (S) announcement

NAME : CW3b.caused_local_event
caused_remote_event1 = (S) none

NAME : CW3b.caused_remote_event1
caused_remote_event2 = (S) clear

NAME : CW3b.caused_remote_event2
dest_line1_state = (S) call in progress

NAME : CW3b.dest_line1_state
dest_line2_state = (S) call in progress

NAME : CW3b.dest_line2_state
feature_name = (S) Call Waiting

NAME : CW3b.feature_name
local_line_state = (S) call in progress

NAME : CW3b.local_line_state
location_attachments = (S) local

NAME : CW3b.location_attachments
rank = (I) Unknown
sequence_number = (I) 3

NAME : CW3b.sequence_number
trigger_event_location = (S) local

NAME : CW3b.trigger_event_location
trigger_event_type = (S) control code

NAME : CW3b.trigger_event_type

Objects describing the *Call divert on no reply* feature

NAME : DNR1
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DNR1.cascade_attachments
caused_local_event = (S) announcement

NAME : DNR1.caused_local_event
caused_remote_event1 = (S) none

NAME : DNR1.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DNR1.caused_remote_event2
dest_line1_state = (S) any

NAME : DNR1.dest_line1_state
dest_line2_state = (S) any

NAME : DNR1.dest_line2_state
feature_name = (S) Call Divert on No Reply

NAME : DNR1.feature_name

local_line_state = (S) idle

NAME : DNR1.local_line_state
location_attachments = (S) local

NAME : DNR1.location_attachments
rank = (I) Unknown
sequence_number = (I) 1

NAME : DNR1.sequence_number
trigger_event_location = (S) local

NAME : DNR1.trigger_event_location
trigger_event_type = (S) control code

NAME : DNR1.trigger_event_type

NAME : DNR2
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DNR2.cascade_attachments
caused_local_event = (S) none

NAME : DNR2.caused_local_event
caused_remote_event1 = (S) seize

NAME : DNR2.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DNR2.caused_remote_event2
dest_line1_state = (S) any

NAME : DNR2.dest_line1_state
dest_line2_state = (S) any

NAME : DNR2.dest_line2_state
feature_name = (S) Call Divert on No Reply

NAME : DNR2.feature_name
local_line_state = (S) ringing no reply

NAME : DNR2.local_line_state
location_attachments = (S) local

NAME : DNR2.location_attachments
rank = (I) Unknown
sequence_number = (I) 2

NAME : DNR2.sequence_number
trigger_event_location = (S) local

NAME : DNR2.trigger_event_location
trigger_event_type = (S) timeout

NAME : DNR2.trigger_event_type

NAME : DNR3
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DNR3.cascade_attachments
caused_local_event = (S) none

NAME : DNR3.caused_local_event
caused_remote_event1 = (S) clear

NAME : DNR3.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DNR3.caused_remote_event2
dest_line1_state = (S) busy

NAME : DNR3.dest_line1_state
dest_line2_state = (S) any

NAME : DNR3.dest_line2_state
feature_name = (S) Call Divert on No Reply

NAME : DNR3.feature_name
local_line_state = (S) any

NAME : DNR3.local_line_state
location_attachments = (S) local

NAME : DNR3.location_attachments
rank = (I) Unknown
sequence_number = (I) 3

NAME : DNR3.sequence_number
trigger_event_location = (S) local

NAME : DNR3.trigger_event_location
trigger_event_type = (S) clear

NAME : DNR3.trigger_event_type

Objects describing the *Call divert on busy* feature

NAME : DOB1
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DOB1.cascade_attachments
caused_local_event = (S) announcement

NAME : DOB1.caused_local_event
caused_remote_event1 = (S) none

NAME : DOB1.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DOB1.caused_remote_event2
dest_line1_state = (S) any

NAME : DOB1.dest_line1_state
dest_line2_state = (S) any

NAME : DOB1.dest_line2_state
feature_name = (S) Call Divert on Busy

NAME : DOB1.feature_name
local_line_state = (S) idle

NAME : DOB1.local_line_state
location_attachments = (S) local

NAME : DOB1.location_attachments
rank = (I) Unknown
sequence_number = (I) 1

NAME : DOB1.sequence_number

trigger_event_location = (S) local

NAME : DOB1.trigger_event_location
trigger_event_type = (S) control code

NAME : DOB1.trigger_event_type

NAME : DOB2
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DOB2.cascade_attachments
caused_local_event = (S) none

NAME : DOB2.caused_local_event
caused_remote_event1 = (S) seize

NAME : DOB2.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DOB2.caused_remote_event2
dest_line1_state = (S) any

NAME : DOB2.dest_line1_state
dest_line2_state = (S) any

NAME : DOB2.dest_line2_state
feature_name = (S) Call Divert on Busy

NAME : DOB2.feature_name
local_line_state = (S) busy

NAME : DOB2.local_line_state
location_attachments = (S) local

NAME : DOB2.location_attachments
rank = (I) Unknown
sequence_number = (I) 2

NAME : DOB2.sequence_number
trigger_event_location = (S) local

NAME : DOB2.trigger_event_location
trigger_event_type = (S) seize

NAME : DOB2.trigger_event_type

NAME : DOB3
CLASSES :
feature_states
PROPERTIES :
cascade_attachments = (S) none

NAME : DOB3.cascade_attachments
caused_local_event = (S) none

NAME : DOB3.caused_local_event
caused_remote_event1 = (S) clear

NAME : DOB3.caused_remote_event1
caused_remote_event2 = (S) none

NAME : DOB3.caused_remote_event2
dest_line1_state = (S) busy

NAME : DOB3.dest_line1_state
dest_line2_state = (S) any

NAME : DOB3.dest_line2_state
feature_name = (S) Call Divert on Busy

NAME : DOB3.feature_name
local_line_state = (S) any

NAME : DOB3.local_line_state
location_attachments = (S) local

NAME : DOB3.location_attachments
rank = (I) Unknown
sequence_number = (I) 3

NAME : DOB3.sequence_number
trigger_event_location = (S) local

NAME : DOB3.trigger_event_location
trigger_event_type = (S) clear

NAME : DOB3.trigger_event_type

Objects describing a new service feature

NAME : next_feature_state

PROPERTIES :

cascade_attachments =	(S) none

NAME : next_feature_state.cascade_attachments

caused_local_event =	(S) Unknown
caused_remote_event1 =	(S) Unknown
caused_remote_event2 =	(S) Unknown
dest_line1_state =	(S) Unknown
dest_line2_state =	(S) Unknown
feature_name =	(S) Unknown
local_line_state =	(S) Unknown
location_attachments =	(S) local

NAME : next_feature_state.location_attachments

sequence_number =	(I) Unknown
trigger_event_location =	(S) Unknown
trigger_event_type =	(S) Unknown

A temporary object

NAME : temp

PROPERTIES :

returnval =	(I) Unknown

Claims

1. A system for detecting interaction between services running on a telecommunications network comprising:

   a computer expert system including:

   a) a data store programmed with data representing attributes of service features;
   b) a rule store programmed with rules which relate feature attributes to interaction behaviours; and
   c) an inference engine which is connected to the data store and to the rule store and which is arranged to process the data and the rules, thereby detecting any interaction between the services.

2. A system according to claim 1, in which the data store includes a plurality of objects including, for each feature which is represented in the data store, a set of objects corresponding to different respective state transitions of the feature.

3. A system according to claim 2, in which each of the said set of objects includes a sequence number corresponding to the position of the respective state transition in the sequence of execution of the feature and at least some of the rules in the rule store reason over the values of the sequence numbers.

4. A system according to any claim 2 or 3, in which the objects are arranged in a hierarchy of superclasses and subclasses of the superclasses, and some of the rules reason over superclasses and others of the rules reason over subclasses.

5. A telecommunications network including a system according to any one of claims 1 to 4.

6. A method of detecting interaction between services running on a telecommunications network comprising:

programming a computer expert system including an inference engine with data representing attributes of service features and with rules relating feature attributes to interaction behaviours; and

7. A method of operating a telecommunications network comprising:

programming a computer expert system with data representing attributes of service features and with rules relating feature attributes to interaction behaviours;
processing the said data and the said rules in an inference engine and detecting thereby any interaction between the said services; and
modifying the operation of the network when any interaction is detected.

8. A method according to claim 6 or 7, in which the said data are stored as a plurality of objects which include, for each feature which is represented, a set of objects corresponding to different respective state transition of the feature.

9. A method according to claim 8, including storing a sequence number for each of the objects in the said set, where the sequence number corresponds to the position of the respective state transition in the sequence of execution of the feature and in which at least some of the rules reason over the values of the sequence numbers.

10. A method according to claim 8 or 9, in which the objects are arranged in a hierarchy of superclasses and subclasses of the superclasses, and some of the rules reason over superclasses and others of the rules reason over subclasses.

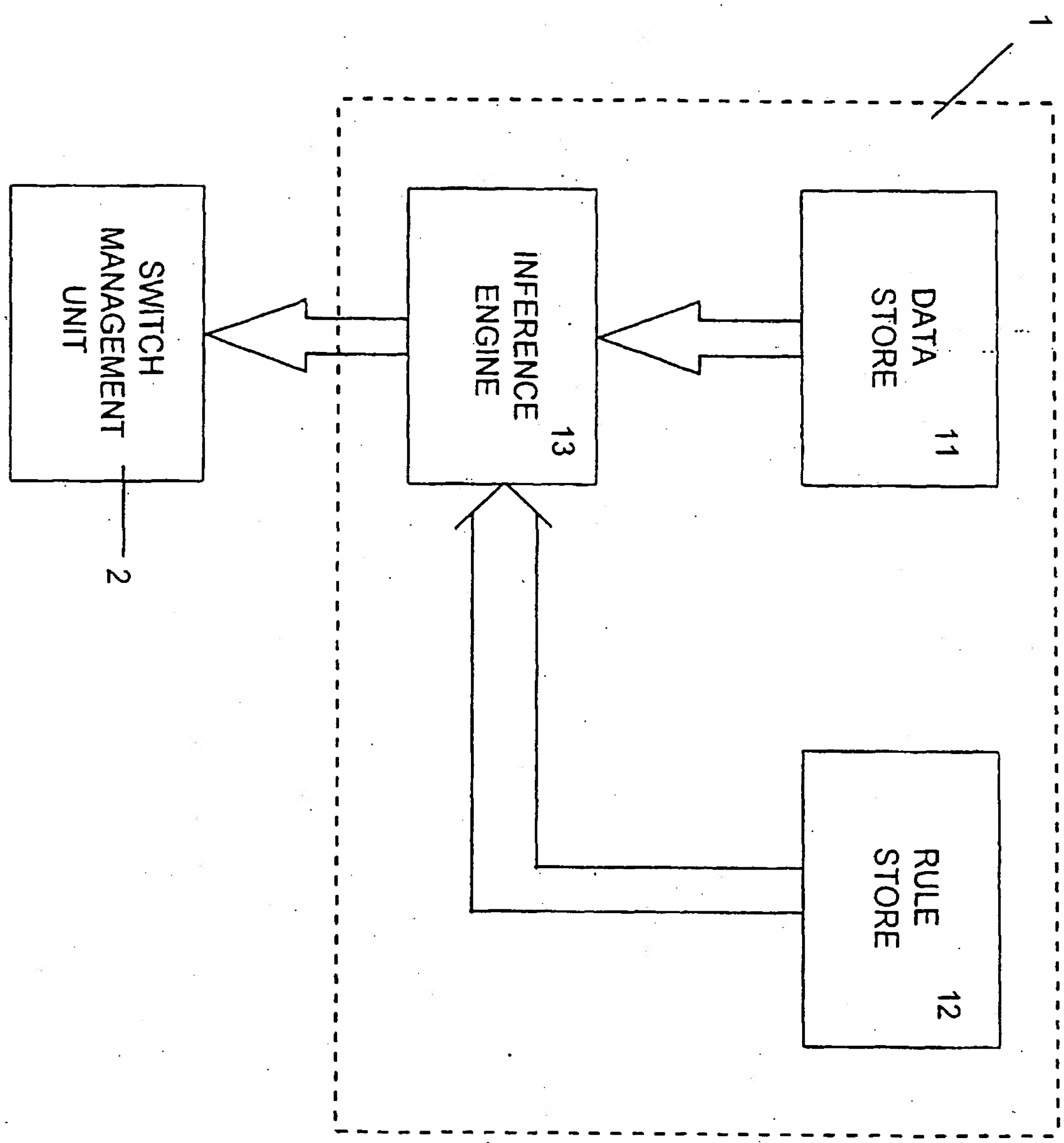

1
DATA STORE
11
RULE STORE
12
INFERENCE ENGINE
13
SWITCH MANAGEMENT UNIT
2

FIGURE 1

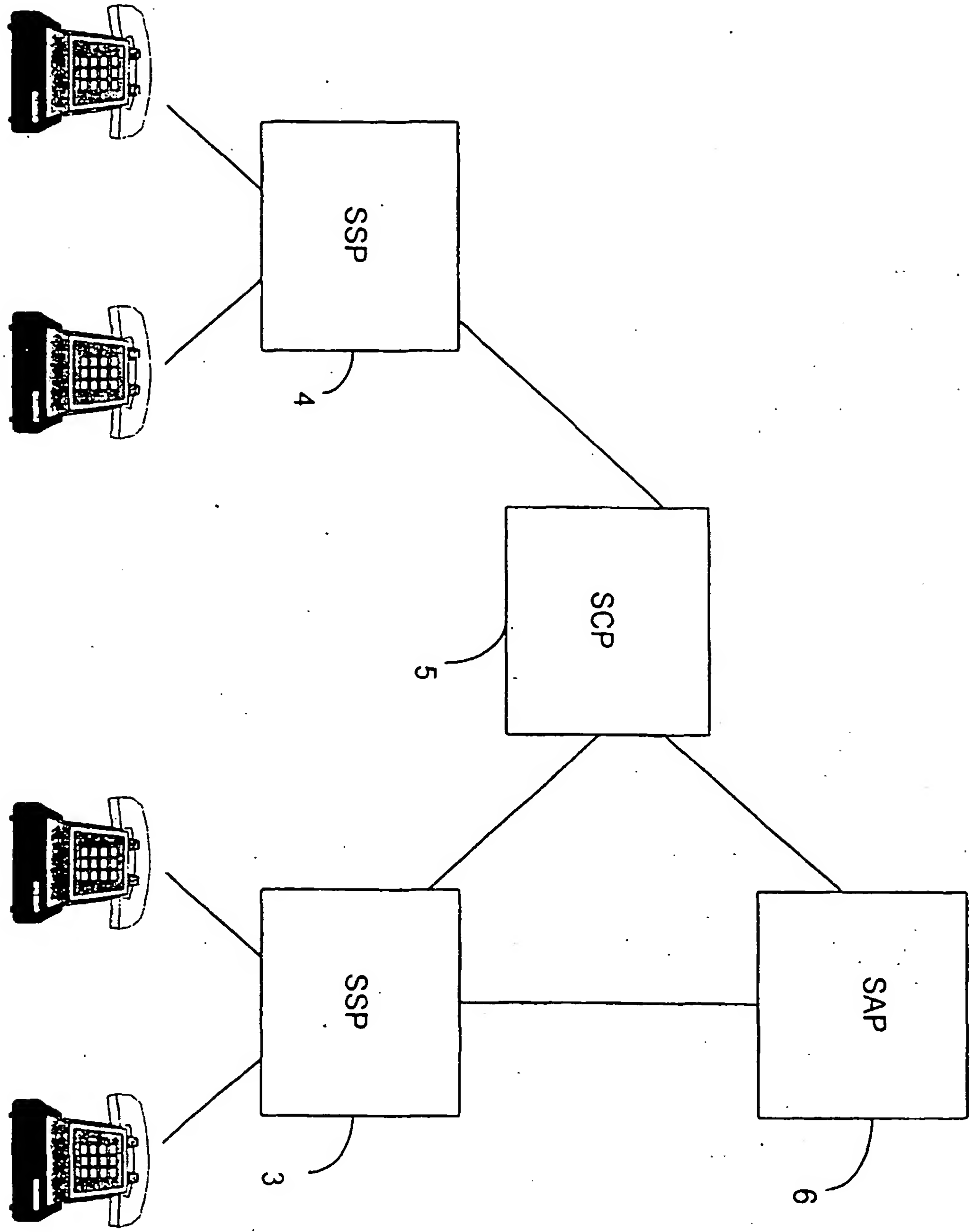


FIGURE 2

FIGURE 3

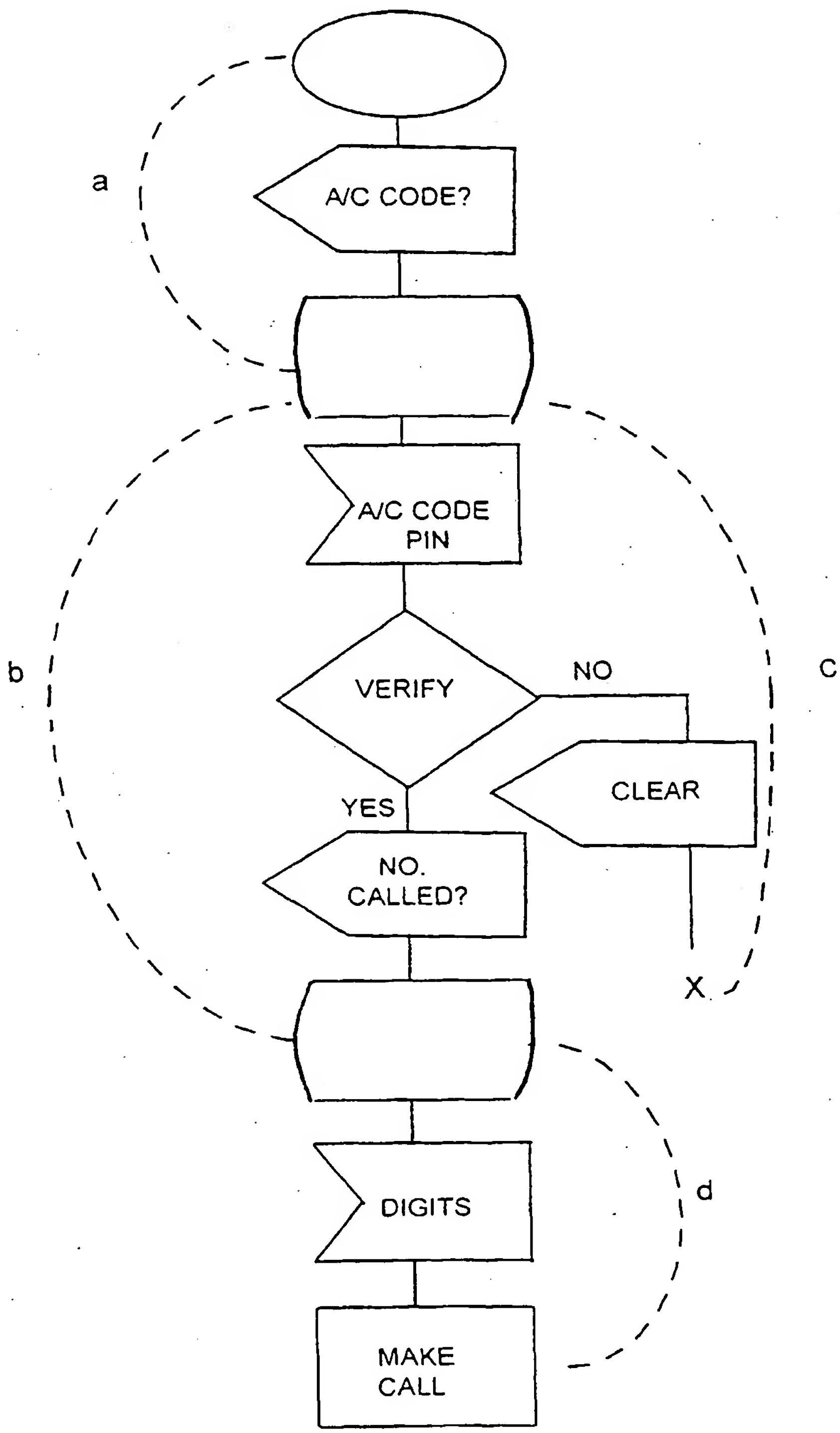
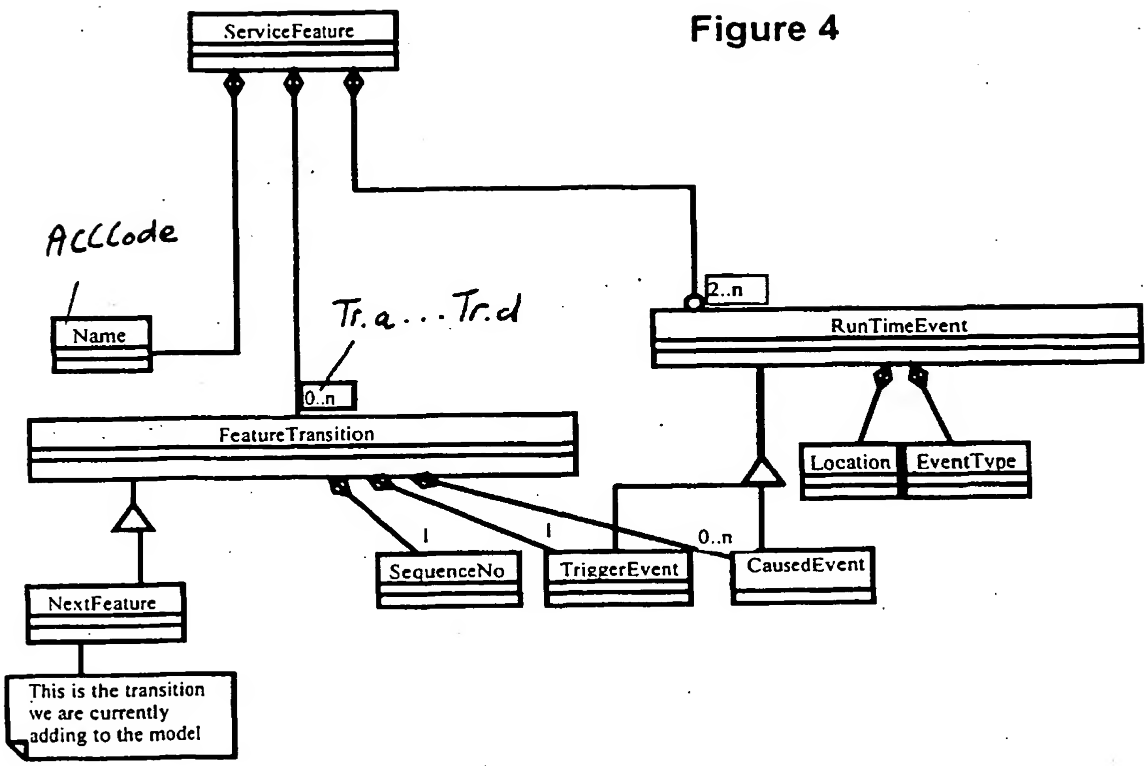

A/C CODE?
A/C CODE
PIN
VERIFY
NO
YES
CLEAR
NO.
CALLED?
X
DIGITS
MAKE
CALL
a
b
c
d

Figure 4


ServiceFeature
AcCCode
Name
Tr.a . . . Tr.d
0..n
FeatureTransition
NextFeature
This is the transition we are currently adding to the model
2..n
RunTimeEvent
Location
EventType
1
1
0..n
SequenceNo
TriggerEvent
CausedEvent

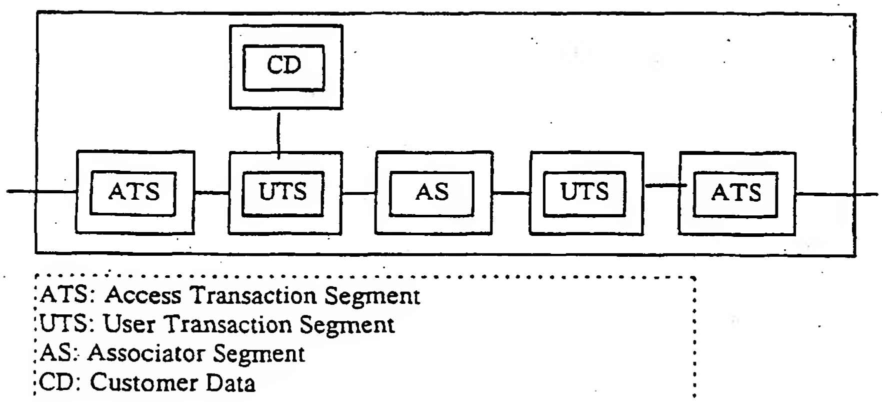

CD
ATS
UTS
AS
UTS
ATS
ATS: Access Transaction Segment
UTS: User Transaction Segment
AS: Associator Segment
CD: Customer Data

Figure 5

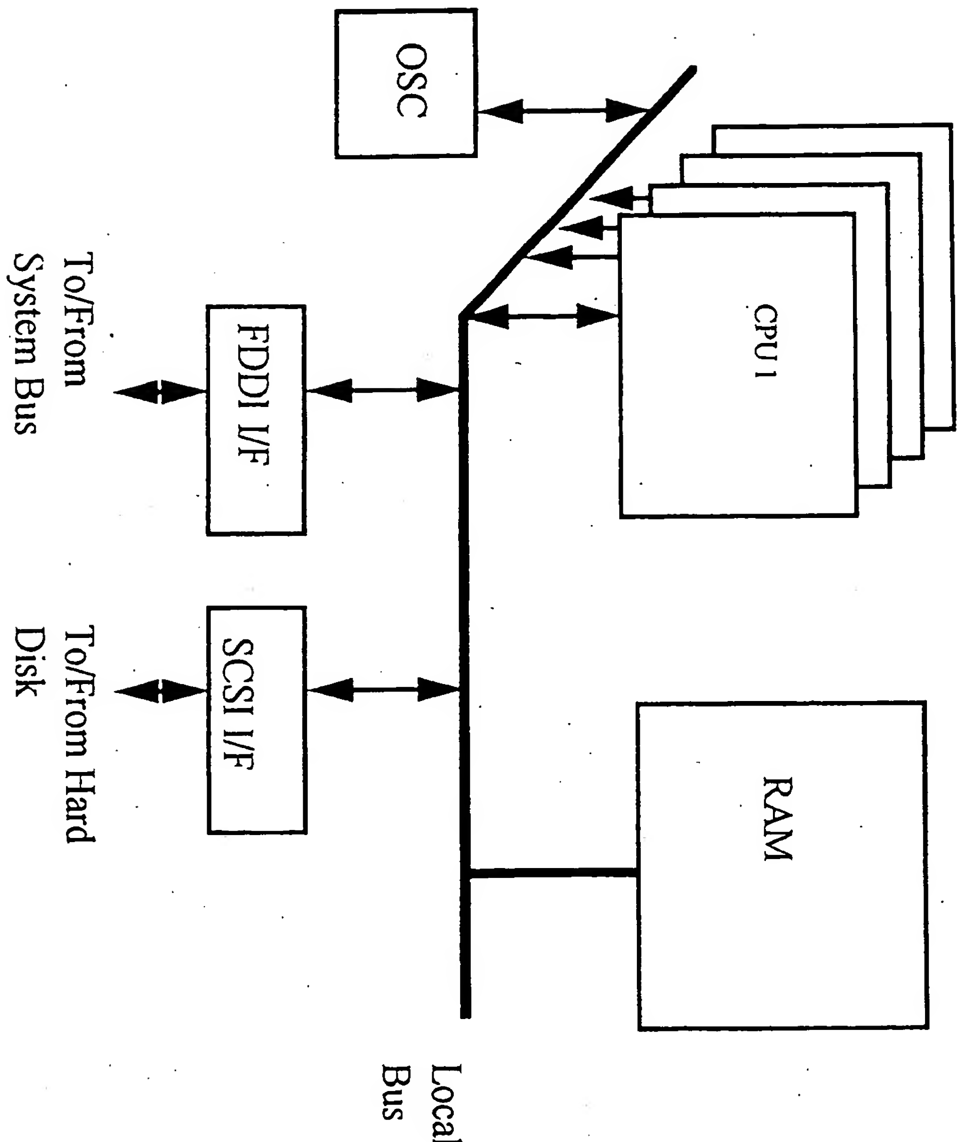

OSC
CPU 1
RAM
FDDI I/F
SCSI I/F
To/From
System Bus
To/From Hard
Disk
Local
Bus

Figure 6

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 7033

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	INTERNATIONAL CONFERENCE ON COMMUNICATIONS, vol. 3, 23 - 26 May 1993, GENEVA CH, pages 1553-1557, XP000448395 BROTHERS ET AL.: "Feature interaction detection" * page 1555, left-hand column, paragraph 3 - page 1556, left-hand column, paragraph 3; figures 2,3 * ---	1,2,5-8	H04Q3/00
A	IEICE TRANSACTIONS ON COMMUNICATIONS, vol. E75-B, no. 10, October 1992, TOKYO JP, pages 986-997, XP000324946 HARADA ET AL.: "A conflict detection support method for telecommunication service descriptions" * page 987, right-hand column, line 12 - page 989, left-hand column, last line * ---	1,2,5-8	
A	WO 93 18606 A (BELL ATLANTIC NETWORK SERVICES) * page 62, line 13 - page 63, line 15 * ---	1,5-7	TECHNICAL FIELDS SEARCHED (Int.Cl.6) H04Q
A	FEATURE INTERACTIONS IN TELECOMMUNICATIONS SYSTEMS, 8 - 10 May 1994, AMSTERDAM NL, pages 36-59, XP000593307 BRAITHWAITE ET AL.: "Towards automated detection of feature interactions" * page 47, line 31 - page 49, line 21 * --- -/--	1,2,5-8	

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	27 March 1997	Lambley, S

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)